Welcome to CS 256:

Algorithm Design and Analysis

About Me

Instructor: Bill Jannen (he/him)

- Reach me:
    - Phone: 597-4509
    - Office: TPL 304
    - Email: 09wkj@williams.edu
- Office hours:
    - Mon 11-noon, Tues 3-4:30 pm, Wed 1:30-3 pm

All at cs.williams.edu/~jannen

Learning and COVID

For a successful course we will have to work together

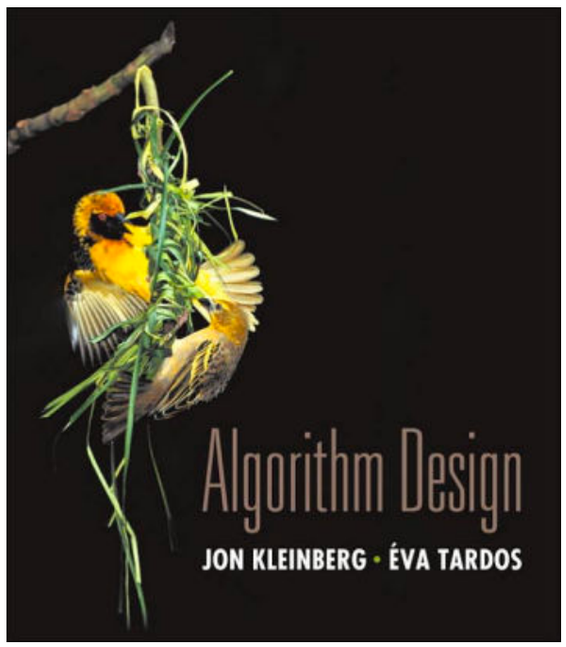
Communicate: if you any concerns or challenges please reach out to me

Participate: in ways that you are comfortable with

- It is OK to wear a mask all semester
- During partner work, respect each others' boundaries
- If you are feeling sick, email me and stay home
    - I will do my best to provide asynchronous* resources

Course Logistics

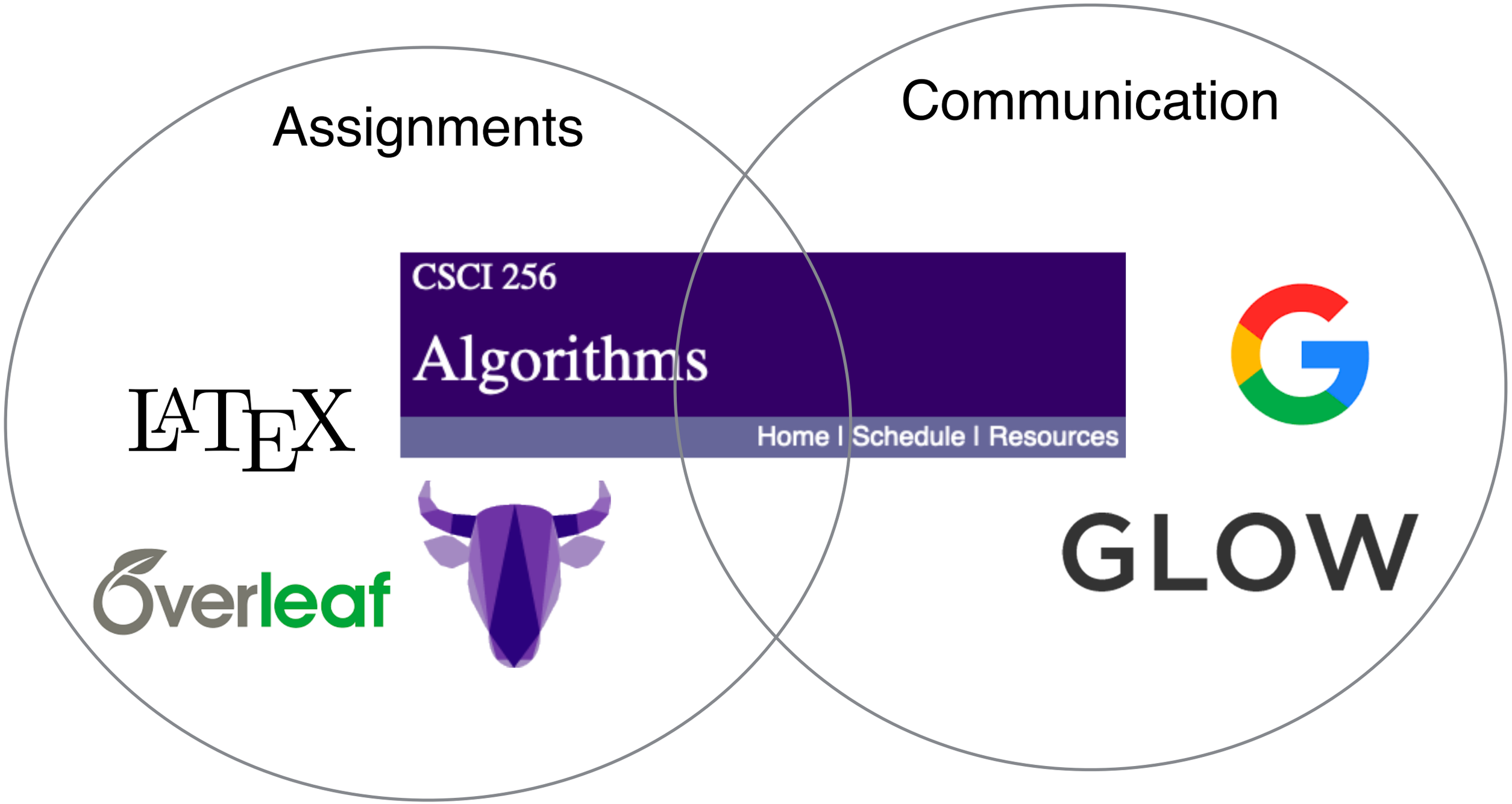
Textbooks:



Several copies reserved in
the Schow library for
reference

Slides: The textbook has excellent slides for reference that I'll also be borrowing from. Feel free to consult them too.

Course Technologies



Course Logistics

Course website:

http://cs.williams.edu/~jannen/teaching/f22/cs256/index.html

Grading breakdown:

- Problem sets (30%)
- Midterm (30%)
    - Take-home 24 hour exam (~10/28)
- Final (30%)
    - Take-home 24 hour exam (final exam period)
- Class participation (10%), includes attendance*
  *Missing class if you are feeling ill is not only acceptable, it is encouraged

About Class Participation

- I would like to have copious amounts of interaction in class!
- Many ways to participate:
  - Ask & answer questions in class
  - Come to office hours
  - Work collaboratively during group activities
- Attendance policy:
  - Required but flexible: safety first (…is what I always say)!
- Class participation does not mean dominating classroom discussions or interrupting your peers

Bottom line: *Help create a vibrant, positive and inclusive classroom environment!*

About Problem Sets

- Must be typeset in LaTeX **using template provided**
- Anonymized grading: No name/ID on homework
- Must be submitted to your GitLab repository

- Assignments will usually be released on Wed and be due the **following Wed at 10 pm**
- Assignment 0 will be out soon! Due **Wed 9/14**

Academic Honesty Policies

- Read collaboration policies on course syllabus
- Gist:
  - Collaboration is encouraged but ***you should never view a written solution that is not your own***
  - External sources okay for background study but you ***cannot search for problem-specific keywords***
  - Always cite your sources and collaborators
    - Cite sources/collaborators in the last section labeled "Acknowledgements" in template
    - Do not miss this part!

Academic Honesty Policies

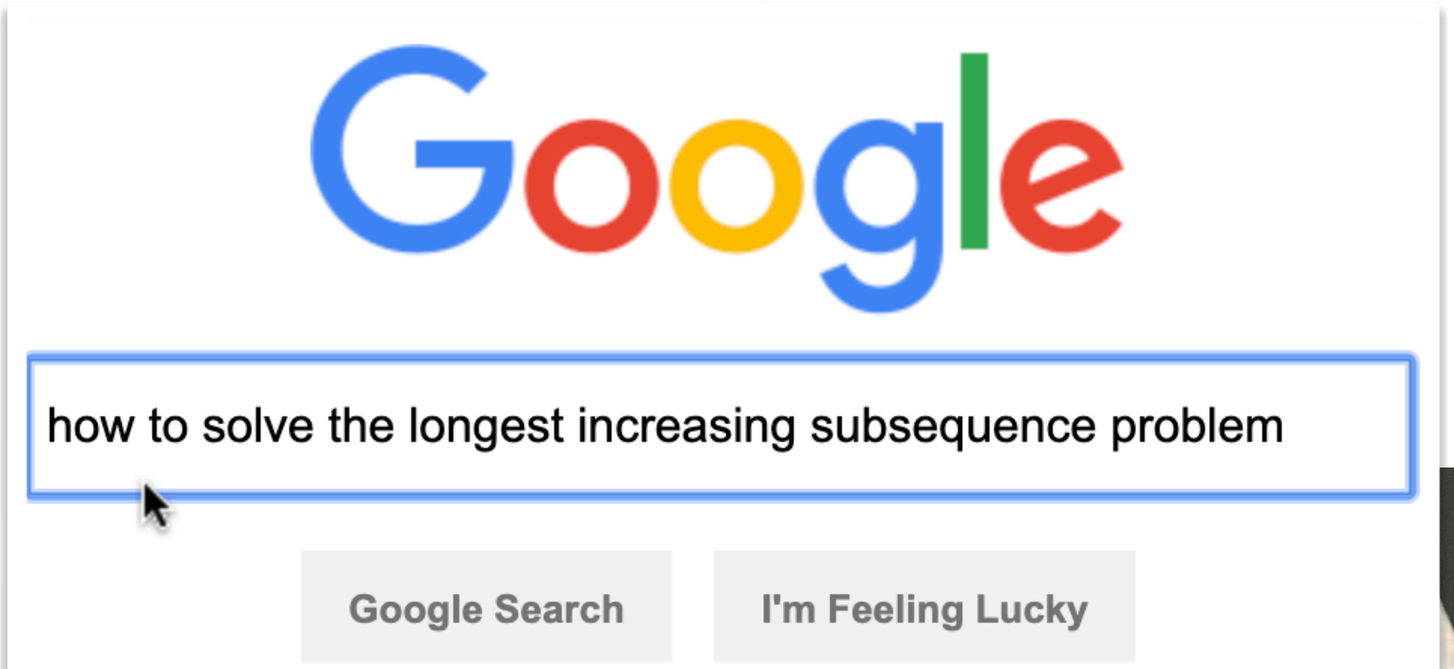
**I didn't full understand dynamic programming in class…
These MIT notes online look good, maybe I will read them to prepare for the assignment**

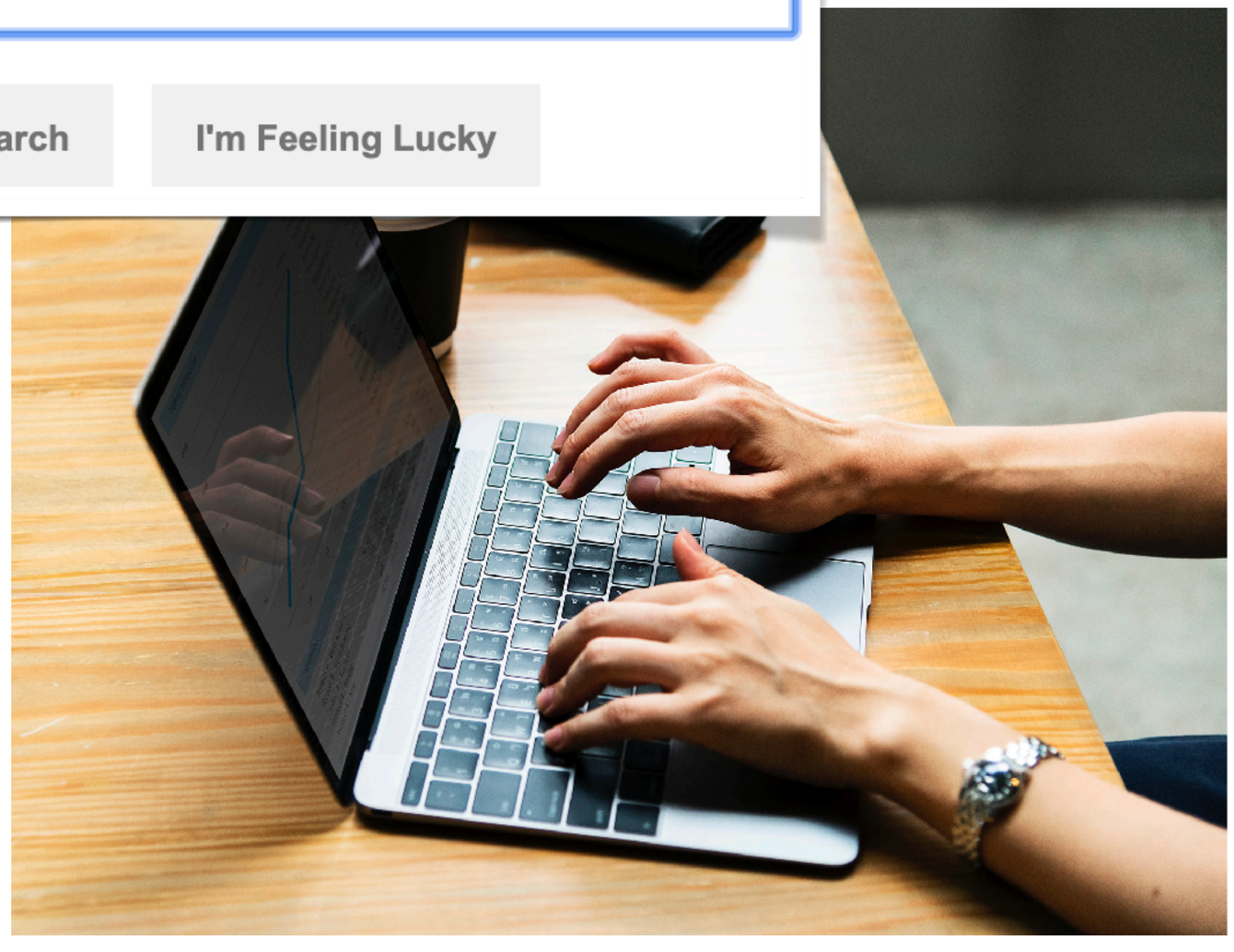
Academic Honesty Policies

**I didn't full understand dynamic programming in class…
These MIT notes online look good, maybe I will read them to prepare for the assignment**

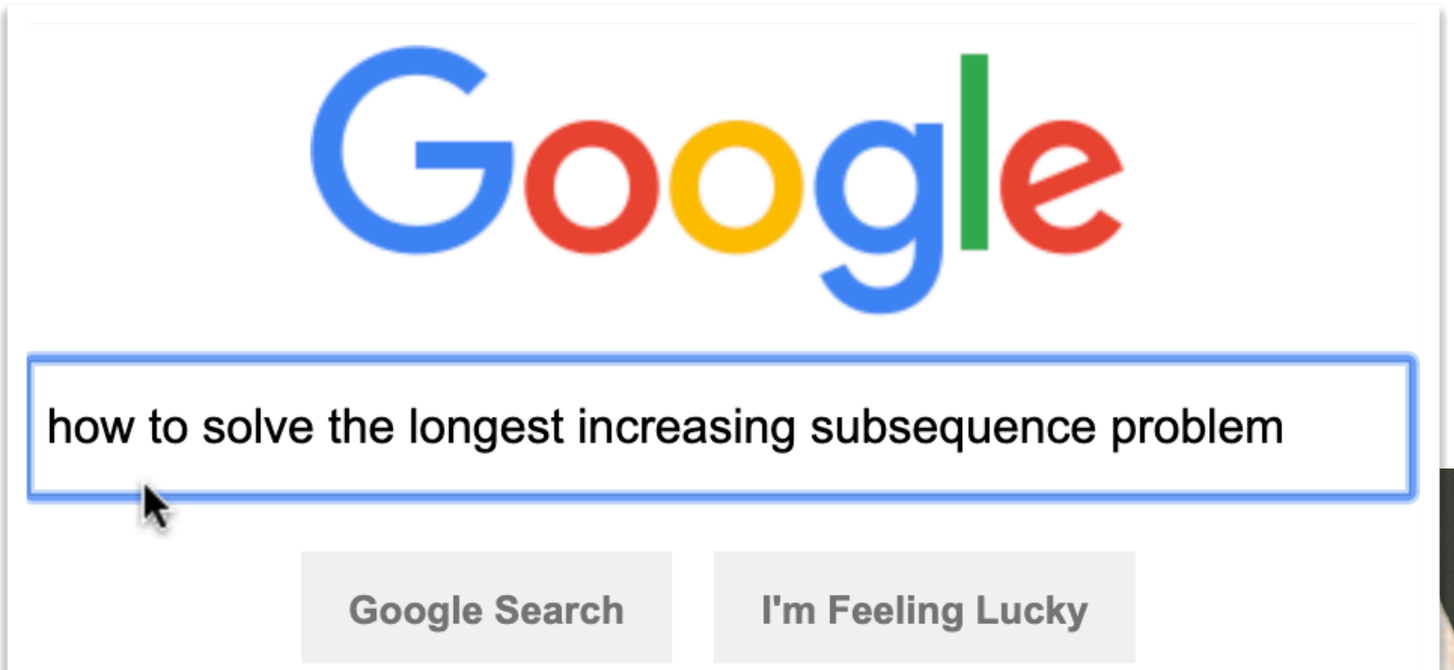
**This Is OK!
(if you cite)**

Academic Honesty Policies

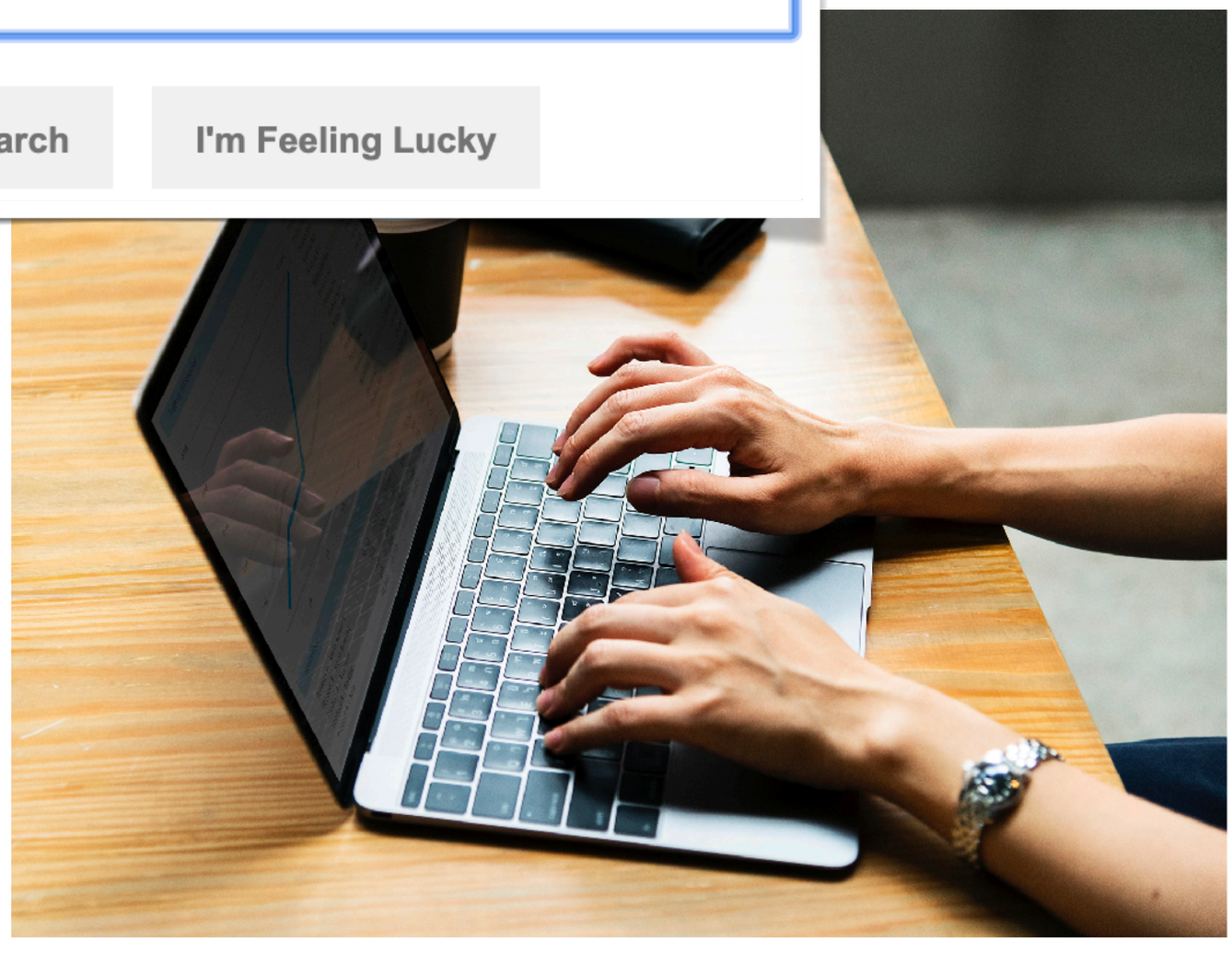




Academic Honesty Policies



This is NOT OK!
(even if you cite)



Academic Honesty Policies

What strategy did you use for Question 3?

I reduced the problem to network flows

https://www.wanderglobe.org/laptops-for-college-students/

Academic Honesty Policies

What strategy did you use for Question 3?

I reduced the problem to network flows

This is OK! (if you cite)

https://www.wanderglobe.org/laptops-for-college-students/

Academic Honesty Policies

Can you show me your solution to Question 3

Sure

https://www.wanderglobe.org/laptops-for-college-students/

Academic Honesty Policies

Can you show me your solution to Question 3

Sure

This is NOT OK!
(even if you cite)

https://www.wanderglobe.org/laptops-for-college-students/

Advice on Collaboration

- Collaborations policies of this course are generous
- What do you see as the benefits of this?
- What do you see as the downsides?
- Problem set advice:
  - HW problems tend to have solutions that require some insight to discover
  - If you immediately start working on the assignments in a group, you will miss out on the opportunity to come up with these insights on your own.
  - Attempting problems yourself first is the single most important practice for the exams

What to Expect from this Class

- Expect **challenging and fun problems**
  - Expect to spend a lot of time thinking about the problems!
  - Sense of accomplishment on finally solving them
- Expect to **make mistakes**
  - Making mistakes is the best way to learn
  - If you knew everything, you wouldn't be in this class
- Expect to **leave your comfort zone**
  - Learning is uncomfortable, but in a good way
  - Common and OK to be frustrated by false starts!
- Expect to develop **"algorithmic thinking"**

Practice with CS Proofs

- Important component of this class (the “Analysis” part of the course name)
- We will learn how to write computer science proofs
    - Slightly different style than mathematics proofs
- Programming assignment vs proofs: common roadblock: how do you know your proof is “correct”?
    - **No autochecker for proofs!** Need to debug yourself
        - Go line by line and ask “why is this true?”
    - Ask me (or TAs) for guidance
    - You’ll build more intuition with practice

How to Succeed

- Read the handout on Problem Set Advice
- Do the readings
    - Ok to skim before class, **read more closely after**
- Go to TA and office hours
    - Treat them as "lab time"
- Form study groups!
    - Both in person and remote
    - I'd be happy help coordinate this

Questions?

Course Overview

- In CS 136, you (likely) learned:
    - How to take large computational problems and break them into small, digestible pieces
    - The asymptotic performance of those pieces can have a very significant impact
- In this class we take this further
    - Learn various algorithm design paradigms
    - How to model and efficiently solve computational problems
    - Analysis and proofs: why does the algorithm work (correctness)? How long does it take (running time)?
    - Study known algorithms and how to use them

Course Outline

- Review of asymptotic complexity
- Graph Traversals: BFS, DFS and applications
- Greedy algorithms
- Divide and conquer (recursion)
- Dynamic programming (recursion without repetition)
- Network-flow algorithms
- Intractability: NP hard, NP completeness & Reductions
- Randomized and approximation algorithms
- …and more!

Stable Matchings

An Illustrative Example:

The Stable Matching Problem

Applications

- Assigning first year students to advisors
- Pairing job candidates with employers
- Matching doctors to hospitals

Fundamental Problem

- Given preferences of both sides, find a **matching** that is resilient against *opportunistic swapping*

State the Problem

- Two groups: hospitals and students
- Students have preferences over hospitals
- Hospitals have preferences over students

Goal. Matching of hospitals to students that is "stable"—that is, no pair has an incentive to break their match

The Redesign of the Matching Market for American Physicians: Some Engineering Aspects of Economic Design

By Alvin E. Roth and Elliott Peranson*

We report on the design of the new clearinghouse adopted by the National Resident Matching Program, which annually fills approximately 20,000 jobs for new physicians. Because the market has complementarities between applicants and between positions, the theory of simple matching markets does not apply directly. However, computational experiments show the theory provides good approximations. Furthermore, the set of stable matchings, and the opportunities for strategic manipulation, are surprisingly small. A new kind of "core convergence" result explains this; that each applicant interviews only a small fraction of available positions is important. We also describe engineering aspects of the design process. (JEL C78, B41, J44)

THE MATCH NATIONAL RESIDENT MATCHING PROGRAM®

National Resident Matching Program

Matching Med-Students to Hospitals

Input. A set H of n hospitals, a
preferences (each hospital ra
students ranks each hospital)

What makes a matching ***good***?
What makes a matching ***bad***?

	1st	**2nd**	**3rd**
MA	Aamir	Beth	Chris
NH	Beth	Aamir	Chris
OH	Aamir	Beth	Chris

	1st	**2nd**	**3rd**
Aamir	NH	MA	OH
Beth	MA	NH	OH
Chris	MA	NH	OH

Perfect Matchings

Definition. A matching M is a set of ordered pairs (h, s) where $h \in H$ and $s \in S$ such that

- Each hospital h is in at most one pair in M
- Each student s is in at most one pair in M

A matching M is **perfect** if each hospital is matched to exactly one student and vice versa (i.e., $|M| = |H| = |S|$)

	1st	**2nd**	**3rd**
MA	Aamir	Beth	Chris
NH	Beth	Aamir	Chris
OH	Aamir	Beth	Chris

	1st	**2nd**	**3rd**
Aamir	NH	MA	OH
Beth	MA	NH	OH
Chris	MA	NH	OH

Unstable Pairs

Definition. A perfect matching M is **unstable** if there exists an unstable pair $(h, s) \in H \times S$, that is,

- h prefers s to its current match in M
- s prefers h to its current match in M

Can you point to any unstable pairings in this matching?

	1st	2nd	3rd
MA	Aamir	Beth	Chris
NH	Beth	Aamir	Chris
OH	Aamir	Beth	Chris

	1st	2nd	3rd
Aamir	NH	MA	OH
Beth	MA	NH	OH
Chris	MA	NH	OH

Unstable Pairs

Definition. A perfect matching M is **unstable** if there exists an unstable pair $(h, s) \in H \times S$, that is,

- h prefers s to its current match in M
- s prefers h to its current match in M

(Beth, MA) are better off together

	1st	2nd	3rd
MA	Aamir	Beth	Chris
NH	Beth	Aamir	Chris
OH	Aamir	Beth	Chris

Can you point to any unstable pairings in this matching?

	1st	2nd	3rd
Aamir	NH	MA	OH
Beth	MA	NH	OH
Chris	MA	NH	OH

Stable Matching Problem

Problem. Given the preference lists of n hospitals and n students, find a stable matching, that is a matching with no unstable pairs.

Question. Does such a matching always exist?

This does not seem obvious!

	1st	**2nd**	**3rd**
MA	Aamir	Beth	Chris
NH	Beth	Aamir	Chris
OH	Aamir	Beth	Chris

	1st	**2nd**	**3rd**
Aamir	NH	MA	OH
Beth	MA	NH	OH
Chris	MA	NH	OH

How Can We Show This?

- Want to prove: a stable matching always exists
- One way:
  - Give an algorithm to find a stable matching
  - Prove that it is always successful
  - Constructive method